by
Jeff Prosise

Tutor

- **NULL-MODEM CABLES: They provide direct PC-to-PC communications, but no standard defines how they should look.**
- **IDENTIFYING MATH COPROCESSORS: Here's how to tell what type of NDP you have.**

NULL-MODEM CABLES

Figure 1 shows two diagrams for null-modem cables: one that appeared in *PC Magazine*'s January 17, 1989, Lab Notes column, the other for a null-modem cable sold by a popular mail-order house. Why are they different? Shouldn't all null-modem cables be similarly wired? And will both cables work with *PC Magazine*'s ZCOPY utility, which also appeared in the January 17, 1989, issue?

William H. Dixon
Leesburg, Florida

Not all null-modem cables are created equal. The Electronics Industries Association (EIA) RS-232 specification from which we derive our standards for asynchronous serial communications doesn't address the issue of direct PC-to-PC communications and therefore doesn't delineate what a null-modem cable should look like. As a result, you'll find several different variations on the "standard" null-modem cable. Whether or not a given cable will work with a selected piece of software depends on how the RS-232 control pins are used to control the flow of data.

In its simplest form, a null-modem cable is nothing more than a serial cable with pins 2 and 3—the Transmit Data (TD) and Receive Data (RD) pins—crossed so that what is transmitted on pin 2 at one end will show up on pin 3 at the other, and vice versa. If no handshaking is performed at the hardware level, this arrangement is sufficient for rudimentary but effective data transfer between devices.

What is hardware handshaking? By definition, hardware handshaking is performed when two programs manipulate RS-232 control pins—DTR, DSR, RTS, and CTS—to achieve a hardware-based form of flow control. For example, in DTR/DSR handshaking, the sender asserts DTR (Data Terminal Ready) before sending the first character in a stream of data and waits for DSR (Data Set Ready) to be asserted in return. RTS/CTS handshaking works in a similar manner, but uses the Request To Send and Clear To Send pins rather than Data Terminal Ready and Data Set Ready. In either case, the sender delays transmitting data until the receiver is ready. Many serial communications programs use these or similar forms of handshaking to ensure that the PC on the transmitting end doesn't attempt to send data before the PC on the receiving end is prepared to accept it.

An alternative to hardware handshaking is software flow control. Programs that use the popular XON/XOFF flow-control protocol, for example, send special XON and XOFF characters (binary codes 17 and 19, respectively) to regulate the flow of data. If the receiver detects an impending buffer-full condition, for example, it transmits an XOFF character signaling the sender to suspend data transmission. When it's prepared to accept more data, it sends an XON character telling the sender to resume. Other alternatives are to forgo

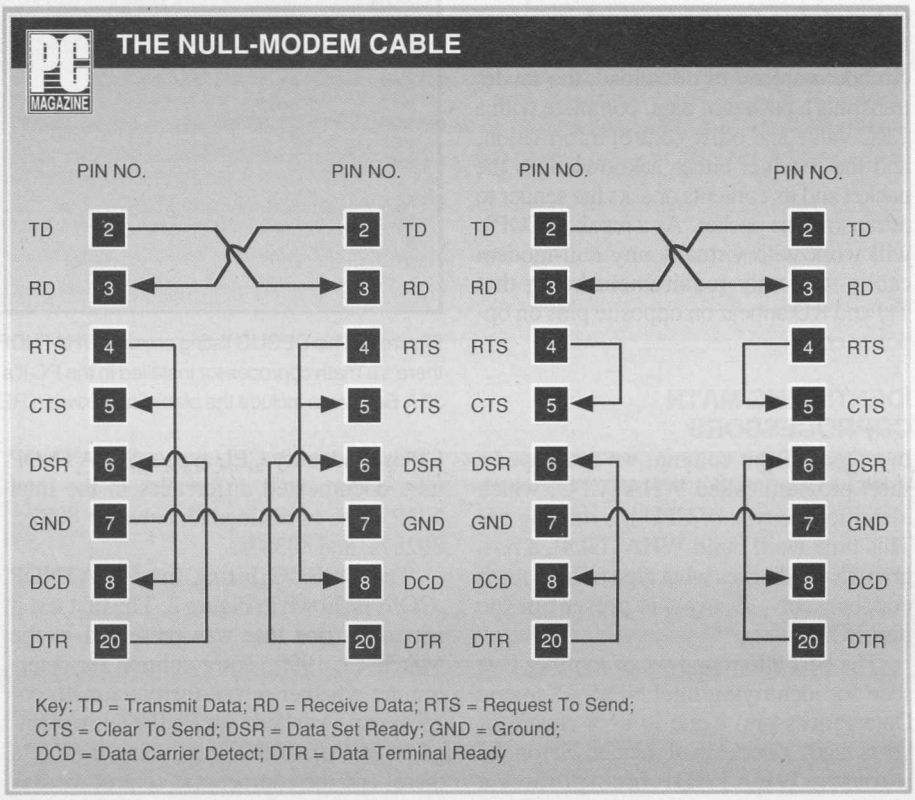

Figure 1: Here are two variations of the "standard" null-modem cable. There is no recognized standard for wiring null-modem cables, because the EIA RS-232 specification does not concern itself with direct PC-to-PC communications issues.

flow control altogether, or to use a packet-based file-transfer protocol such as Kermit or Xmodem.

The advantage to using a file-transfer protocol is that it's relatively easy to build-in error-checking logic, ensuring not only that the sending and receiving devices communicate with each other, but also that what comes across the line is free from random errors.

A typical null-modem cable is wired to short-circuit a program's dependency on any RS-232 pins other than TD and RD. Both cables shown in Figure 1, for example, work equally well with programs that employ DTR/DSR handshaking, RTS/CTS handshaking, or no handshaking at all. In addition, both cables tie an RS-232 output into DCD just in case the software running on the PC won't work unless Data Carrier Detect is asserted. The only difference is the physical wiring of DCD: the cable on the left loops the RTS output into DCD, while the one on the right loops DTR into DCD.

Will these null-modem cables work with ZCOPY? Yes. ZCOPY uses a packet-based data-transfer protocol in lieu of handshaking. The transfer isn't unlike an Xmodem upload or download; the sender transmits a packet of data, complete with a CRC value and other control information, and the receiver either acknowledges the packet and its contents or asks the sender to retransmit the packet. As a result, ZCOPY will work with virtually any null-modem cable—the only requirement being that TD and RD appear on opposite pins on opposite ends.

WHATNDP.SCR **COMPLETE LISTING**

```
N WHATNDP.COM
A
JMP       0136
DB        0D,0A,"No NDP installed",0D,0A,"$"
DB        0D,0A,"NDP is an 80$"
DB        "87$"
DB        "287$"
DB        "387$"
DB        0D,0A,"$"
DB        0,0
FINIT                              ;Check for coprocessor
FSTCW     [0134]                   ;  by initializing it
CMP       BYTE PTR [0135],3        ;  and looking at the
JZ        014B                     ;  bits 8 and 9 of the
MOV       AH,9                     ;  control word
MOV       DX,0102                  ;Exit if there's nothing
INT       21                       ;  there
RET
MOV       AH,9                     ;Print opening message
MOV       DX,0117
INT       21
MOV       DX,0126                  ;Test for 8087 by seeing
AND       WORD PTR [0134],FF7F     ;  if the FDISI instruction
WAIT                               ;  sets the IEM bit in the
FLDCW     [0134]                   ;  coprocessor control word
WAIT
FDISI
WAIT
FSTCW     [0134]
TEST      WORD PTR [0134],80
JNZ       018F                     ;Branch if NDP is an 8087
MOV       DX,0129                  ;Distinguish between 287
FINIT                              ;  and 387 by determining
FLD1                               ;  whether the NDP is
FLDZ                               ;  initialized to affine
FDIVP     ST(1),ST                 ;  (387) or projective
FLD       ST(0)                    ;  (287) infinity
FCHS
FCOMPP
FSTSW     [0134]
WAIT
MOV       AX,[0134]
SAHF
JZ        018F                     ;Branch if 287
MOV       DX,012D                  ;It's a 387!
MOV       AH,9                     ;Print NDP type and exit
INT       21
MOV       AH,9
MOV       DX,0131
INT       21
RET

RCX
9B
W
Q
```

Figure 2: This DEBUG listing creates WHATNDP.COM, a short utility that reveals whether or not there's a math coprocessor installed in the PC it's run on, and if there is, whether it's an 8087, 287, or 387. Be sure to include the blank line between RET and RCX near the end of the listing.

IDENTIFYING MATH COPROCESSORS

In a recent Tutor column, we developed a short program called WHATCPU, which identifies the type of CPU it's running on. This time we'll build WHATNDP, a program that identifies what type of Intel math coprocessor—if any—is present in the host PC.

The best all-around set of routines I've seen for identifying Intel NDPs (Numeric Data Processors) came to us a couple of years ago, courtesy of reader Baron L. Roberts. WHATNDP incorporates a slightly modified version of his routines. Like WHATCPU, which relies on known differences between the various Intel CPUs to identify CPU types, WHATNDP uses documented differences in the Intel NDP chips to distinguish between 8087s, 80287s, and 80387s.

The DEBUG listing for WHATNDP.COM is shown in Figure 2. The first test it applies is one that was presented in the March 13, 1990, Tutor column for determining whether or not there's a math coprocessor installed. An FNINIT is executed to initialize the coprocessor (if it's there) and the coprocessor control word is transferred to a memory location where it can be inspected. FNINIT sets bits 8 and 9 of the control word on all Intel math chips. Thus, if bits 8 and 9 of the memory location that the control word was transferred to are set after the operation is complete (the memory location was originally set to 0), WHATNDP concludes that there must be a coprocessor present.

The next test separates 8087s from 287s and 387s. The key is that the coprocessor instructions FDISI and FENI, which disable and enable coprocessor interrupts by setting and clearing the IEM (Interrupt Enable Mask) bit in the coprocessor control word, are ignored on the 287 and 387 but not on the 8087. WHATNDP manually clears the IEM bit, then executes an FDISI